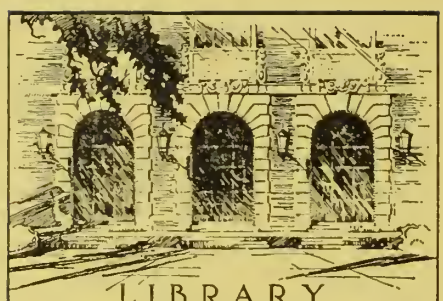

Report No. 276

ON THE GENERATION OF PARSERS FOR BNF GRAMMARS: AN ALGORITHM

by

Franklin L. DeRemer

ILLIAC IV Document No. 199

DEPARTMENT OF COMPUTER SCIENCE · UNIVERSITY OF ILLINOIS · URBANA, ILLINOIS

ILLIAC IV Document No. 199

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS 61801

Contract No.
US AF 30 (602)4144

ON THE GENERATION OF PARSERS FOR BNF GRAMMARS: AN ALGORITHM

by

Franklin L. DeRemer

Report No. 276
August 1, 1968

This work was supported in part by the Department of Computer Science, University of Illinois, Urbana, Illinois, and in part by the Advanced Research Projects Agency as administered by the Rome Air Development Center, under Contract No. US AF 30(602)4144.

ACKNOWLEDGEMENT

The author would like to thank Alan J. Beals for his help in evaluating and debugging the algorithm. Thanks are also due Dr. R. S. Northcote for suggested improvements in the paper itself.

ABSTRACT

This paper describes an algorithm which, for suitable grammars, maps the Backus Naur Form (BNF) definition of the grammar of a language into a parser for the sentences in that language. By design the algorithm generates a suitable parser for any bounded right context grammar. It happens that it also covers some LR(k) grammars which are not bounded right context. A modified version of Floyd's descriptive language for symbol manipulation is used to describe the parser. Several examples illustrate the application and generality of the algorithm.

Introduction

The algorithm described herein is in essence an extension, albeit a simplification, of the work of Earley[1] which in turn was based on Evans,[2] Feldman,[3] Floyd,[4,5] and Standish[7]. For a large subset of grammars, the algorithm maps the Backus Naur Form (BNF) definition of the grammar of a language into a deterministic, left-to-right parser for the sentences in that language. It is shown below that the algorithm, by design, covers all bounded right context grammars and, as a by-product, some LR(k) grammars which are not bounded right context (see Knuth[6] for the definitions of these classes of grammars).

More precisely, the algorithm maps a set of BNF productions into a program: a "reductions analysis" program* consisting of modified Floyd productions (really reductions) referred to below as FPL (Floyd Production Language) statements.** The program consists of labeled, mutually exclusive groups of statements called sections. Each section has a specific task to perform. It is activated, by transfer of control to its first statement via the label, only at appropriate times. Upon each activation it either scans a new terminal symbol or makes a reduction (combined with an "unscan" in the case of a production with an empty right part) and then transfers control to the appropriate next section, or it transfers control to an ERROR routine if control falls out the bottom of the section.

The algorithm is based on Earley's intuitive notion that the top symbols on the stack matched against the right parts of certain productions should determine parsing decisions. It is an extension of his algorithm in that it provides for both finite look-ahead and finite look-back and in that it covers productions with empty right parts. It is a simplification of his alogrithm in that it allows reductions only at the top of the stack, therefore reducing the number of mapping rules.

---

*It is assumed that the reader is familiar with reductions analysis programs and the associated stack, input string, and manipulations thereupon.

**This nomenclature is adapted to clarify the distinction between the BNF productions, which together define the grammar of a language, and the FPL statements which, when combined to form a program, describe a parser.

A word notation is in order before proceeding. In this paper, non-terminal symbols are represented by Latin capitals, terminals by lower case Latin letters, arbitrary strings by the Greek letter α, and the empty string by the Greek letter ϵ. The Greek letter σ designates a symbol which matches any other symbol.

The Algorithm

The algorithm simply consists of : (a) three rules to determine what sections are necessary for the program, (b) three corresponding rules to determine which productions should be mapped into statements for each section, (c) four rules to map the productions into statements, (d) a rule which prescibes the combination of some statements in a given section and a corresponding combination of certain sections, and (e) a contextual analysis rule for expanding statements so no two statements in a given section are both applicable to a given stack and input string configuration. (The latter operation is referred to below as making the statements disjoint.) Of course, there are also several rules for optimizations, some of which are given toward the end of the paper.

(a) Necessary Sections. A special START section and a special section for SUCCESS EXIT are required together with the following:

(1) A section labeled Nh is required for each non-terminal N which appears in the right part of some production as other than the first symbol. This section is activated whenever one of the terminals, which may begin N, is supposed to be at the top of the stack. It is the purpose of section Nh to verify that one of these terminal "head symbols" is indeed at the top and, depending upon which terminal is there, to take appropriate action to commence, and perhaps conclude, a reduction to N.

(2) A section labeled $t(\pi,p)$ is required for each occurrence of a terminal as the p-th symbol in the right part of each production π, where $p \geq 2$. This section consists of exactly one statement which compares the first p symbols of production π with the top p symbols of the stack. It is activated only when the match must occur for a well-formed string. Its purpose is to verify the top symbol and to take appropriate action to continue the parse.

(3) A section labeled Nt is required for each non-terminal N which appears in the right part of some production. The section is activated immediately after a reduction to N occurs at the top of the stack. The statements in this section indicate comparisons to the stack to determine which of the production(s) in whose right part N appears is applicable to the case at hand. A match determines the appropriate subsequent action.

(b) Descripton Sets. In order to generate the appropriate set of statements for a given section, a descriptor set of pairs $D = ((\pi_1, p_1),\ldots)$ is assocated with each section label. This descriptor set is determined by investigating the productions and serves to indicate to which part of which production(s) the mapping rules described below are to be applied. The pair (π,p) points to the first p symbols of production π as the stack comparison symbols of the corresponding statement. The descriptor sets are determined as follows:

(1) D_{Nh}: Initially D_{Nh} is empty and the following recursive procedure is applied. The right part of each production π that defines N is examined. If it is empty, then $(\pi,0)$ is added to D_{Nh}; if it begins with a terminal, then $(\pi,1)$ is added to D_{Nh}; otherwise it begins with a non-terminal and the procedure is applied to that non-terminal.

(2) $D_{t(\pi,p)}$ contains exactly one pair (π,p).

(3) D_{Nt}: The right part of each production π is examined. If the non-terminal N appears as the p-th symbol, then (π,p) is in D_{Nt}.

(c) The BNF to FPL Mapping Rules. Presented in Table I are four rules for mapping BNF productions into FPL statements. Together with the descriptor sets they represent a naive first try at generating a parser for the grammar. Implicity, the rules assume there is no question about which production applies to the case at hand but only what action is to be taken by the parser next, given that a certain production is applicable.

Table I

The BNF to FPL mapping rules. (α represents the first p symbols of production π, σ is a symbol which matches any other symbol and q = p + 1.)

BNF production (π,p)	maps into	FPL statement
(1) M ::= αN ...	⟹	α| * Nh
(2) M ::= αb ...	⟹	α| * t(π,q)
(3) M ::= α	⟹	α| → M| Mt
(4) M ::= ϵ	⟹	σ| → M| σ Mt

It is the purpose of the last two rules of the algorithm to extend it to cover a reasonable set of grammars by resolving confusion about which production(s) may apply to different cases within a given section.

The rules of Table I are explained intuitively as follows. If the first p symbols of the right part of production π are at the top of the stack and

(1) if the (p+1)st symbol is a non-terminal N, then the parser should scan(*) the next terminal and activate section Nh to begin to reduce a substring to N.

(2) if the (p+1)st symbol is a terminal b, then the parser should scan the next terminal and activate section $t(\pi,q)$, where $q = p + 1$, to verify that that terminal is indeed b and to decide how to continue the parse.

(3) if the p-th symbol is last in the right part of the production, then the parser should make a reduction ($\rightarrow$) to the symbol M defined by the production and activate section Mt to decide how to continue the parse.

(4) if $p = 0$ (and, therefore, the right part of the production is empty), then the parser should "unscan" the top symbol, push an M onto the top of the stack, and activate section Mt to decide how to continue the parse. (The symbol unscanned will always be a terminal since this statement will appear only in an Nh-type section, the activation of which is always immediately preceded by a scan (see rule (1)).)

(d) Combinations. In general, a reductions analysis program generated according to the above rules will contain sections in which some of the statements are not disjoint. That is, the conflicting statements will indicate stack comparisons (1) which are identical, or (2) the shorter of which are identical to the top few symbols of the longer ones. Thus, several statements may be applicable to a single stack and input string configuration, and the parser is in some sense non-deterministic. To render the parser deterministic it must be modified so it can either delay

or determine the decisions concerning which of the several similar productions associated with the conflicting statements is applicable in various cases. Decision delays are effective by pairwise statement combinations as follows.

> If a pair of statements in a given section are not disjoint and if each was generated according to either mapping rule (1) or (2), then replace them with a single statement: one whose stack comparison is the shorter of the two and which, upon a successful stack match, scans a new terminal and activates a new combination section which must be added to the program. The new section is that section whose description set is the union of the two descriptor sets of the sections which the original statements would have activated. Of course, the new section must be checked for disjointness, and the old sections, of which the new one is a combination, should be checked for usefulness, since the only reference in the entire program to one or both might have been deleted by removal of the two statements.

(e) Expansion by Contextual Analysis. The only decisions which cannot be delayed are those concerning reductions. This limitation is due to the requirement that reductions be made only at the top of the stack. Thus, conflicts with statements generated according to mapping rules (3) and (4) cannot be cured by combination. In this case the statements' comparison fields are expanded by contextual analysis to provide the parser with whatever finite look-ahead and look-back are necessary to make the decision at hand; i.e.,

> for each of the conflicting statements the grammar is investigated and generation begun of the strings of symbols which, in the context of the production associated with the statement, may surround the original stack comparison substring α of the statement. Appropriate comparison of the composite strings associated with each of the original statements, indicates the minimum context which must be checked to make the statements disjoint. In the worst case each statement must be replaced

with several statements which differ from the original in that they indicate more symbols which must be matched in the stack and/or the input string.

Examples

Since the parser proceeds from left to right, always making reductions at the top of the stack on the basis of whatever finite look-ahead and look-back are necessary, the algorithm by definition covers all bounded right context grammars. Further, due to the fact the sections of the program themselves imply certain extra information about the stack configuration, in the same sense that a state of a finite state acceptor implies information about the string read, the algorithm also covers some LR(k) grammars which are not bounded right context. An example grammar in this class is S ::= aA|bB, A ::= cA|d, B ::= cB|d, the sentences of which are a c^n d and b c^n d. It is not bounded right context since the clue as to whether to reduce d to A or B is an a or b arbitrarily far down the stack. The grammar is however, LR(0) and can be parsed by the algorithmically generated parser of Figure 1. Note that a transfer of control to an ERROR routine is implicit at the bottom of each section in case no match occurs.

START(Sh)	a|	*		Ah
	b|	*		Bh
Ah	c|	*		Ah
	d|	→	A|	At
Bh	c|	*		Bh
	d|	→	B|	Bt
At	cA|	→	A|	At
	aA|	→	S|	St
Bt	cB|	→	B|	Bt
	bB|	→	S|	St
St				SUCCESS EXIT

Figure 1. Algorithmically generated parser for a grammar which is LR(0) but not bounded right context.

As an example of a grammar requiring both look-ahead and look-back consider the following.

π		p
		123
1	S ::=	cAB
2	S ::=	dAe
3	A ::=	aG
4	B ::=	xe
5	G ::=	Gx
6	G ::=	x

Confusion arises in the Gt section about when to terminate the gathering of x's into the non-terminal G. Generation of the context related to production five produces three possible strings:

(1) G| → G|x → G|xx
(2) G| → G|x → aG|x → daG|xe
(3) G| → G|x → aG|x → caG|xB → caG|xxe

There are two possible strings for production three:

(1) aG| → daG|e
(2) aG| → caG|B → caG|xe

Most of the confusion is between case (2) of production five and case (2) of production three. One possible solution is to construct the following Gt section:

Gt	G|xx	*		t(5,2)
	daG|xe	*		t(5,2)
	aG|	→	A|	At

Note that advantage has been taken of the sequential nature of the program here. Since the first two statements will catch all configurations to which production five is applicable, the statement associated with production three checks no extra context. That is, the restriction that the statements in a given section must be disjoint may be relaxed in special cases where advantage is taken of the order in which statements are executed, however the contextual analysis must still be performed to ascertain the validity of such an optimization. Finally, note that had production five been G ::= xG the grammar would not have been bounded right context nor covered by this algorithm, although it would still be LR(2).

As a final, larger, and more practical example consider the grammar of Table II, which is Earley's example of a simple algebraic language. The corresponding list of necessary sections and their descriptor sets are presented in Table III, and the parser is given in Figure 2. This grammar requires no special look-back and look-ahead of more than one symbol in only one case, section Dt. A single pair of statements were combined in section Ht causing the combination of sections t(12,2) and t(4,2) to form a section labeled t(1,2; 4,2). Note that such combinations are probably most efficiently effected by operations on the descriptor sets before the sections are generated. Also note that maximum advantage was taken of ordering the statements. However, for expositional purposes several optimizations were not made: (1) since the first p-1 symbols are matched immediately prior to its activation, a t(π,p) section need match only the p-th symbol with the top symbol of the stack, (2) since a reduction to N occurs immediately prior to the activation of an Nt section, it need not match the top symbol, and (3) several sections could have been "concatenated", as for example sections Dt, t(6,2), and t(6,3) which would form

Dt	D|;r	***		T_{ℓ}h
	bD|		H|	Ht

Finally, since sections Ph, Fh, and Th are identical, and are a subset of section Eh, all these could have been combined to save space; however this is probably undesireable as it implies a loss of information useful for error recovery.

PRODUCTION TABLE

				p			
	π	0		1	2	3	4
<AXIOM>	0	Σ	::=	|-	B	-|	
<BLOCK>	1	B	::=	H	e		
<HEAD>	2	H	::=	b			
	3	H	::=	b	D		
	4	H	::=	H	;	S	
<DECLARATION>	5	D	::=	r	T_ℓ		
	6	D	::=	D	;	r	T_ℓ
<TYPE LIST>	7	T_ℓ	::=	i			
	8	T_ℓ	::=	T_ℓ	,	i	
<STATEMENT>	9	S	::=	i	←	E	
<EXPRESSION>	10	E	::=	T			
	11	E	::=	$\pm$	T		
	12	E	::=	E	$\pm$	T	
<TERM>	13	T	::=	F			
	14	T	::=	T	*	F	
<FACTOR>	15	F	::=	P			
	16	F	::=	F	↑	P	
<PRIMARY>	17	P	::=	i			
	18	P	::=	(	E	)	

NOTE: i is identifier
r is real
b is begin
e is end

Table II. Production table for a simple algebraic language.

NECESSARY SECTIONS	DESCRIPTOR SETS			
S T A R T (Σh)	0,1			
B h	2,1	3,1		
D h	5,1			
T_ℓh	7,1			
T h	17,1	18,1		
E h	11,1	17,1	18,1	
S h	9,1			
F h	17,1	18,1		
P h	17,1	18,1		
t (1,2) } combined to form t (1,2; 4,2)				
t (4,2) }				
t (6,2)				
t (8,2)				
t (9,2)				
t (12,2)				
t (14,2)				
t (16,2)				
t (0,3)				
t (6,3)				
t (8,3)				
t (18,3)				
H t	1,1	4,1	(combination)	
D t	6,1	3,2		
T_ℓt	8,1	5,2	6,4	
T t	10,1	14,1	11,2	12,3
E t	12,1	18,2	9,3	
F t	13,1	16,1	14,3	
P t	15,1	16,3		
B t	0,2			
S t	4,3			
Σ t	---			

Table III. List of necessary sections and the corresponding descriptor sets for the grammar of Table II.

S T A R T (Σh)	|-|	*		B h
B h	b|r	*		D h
	b|	→	H|	H t
D h	r|	*		T_ℓh
T_ℓh	i|	→	T_ℓ|	T_ℓt
T h	i|	→	P|	P t
	(|	*		E h
E h	$\pm$|	*		T h
	i|	→	P|	P t
	(|	*		E h
S h	i|	*		t (9,2)
F h	i|	→	P|	P t
	(|	*		E h
P h	i|	→	P|	P t
	(|	*		E h
t (1,2; 4,2)	He|	→	B|	B t
	H;|	*		S h
t (6,2)	D;|	*		t (6,3)
t (8,2)	T_ℓ,|	*		t (8,3)
t (9,2)	i←|	*		E h
t (12,2)	E$\pm$|	*		T h
t (14,2)	T*|	*		F h
t (16,2)	F↑|	*		P h
t (0,3)	|-B-||	→	Σ|	Σ t
t (6,3)	D;r|	*		T_ℓh
t (8,3)	T_ℓ,i|	→	T_ℓ|	T_ℓt
t (18,3)	(E)|	→	P|	P t
H t	H|	*		t (1,2; 4,2)
D t	D|;r	*		t (6,2)
	bD|	→	H|	H t
T_ℓt	T_ℓ|,	*		t (8,2)
	D;rT_ℓ|	→	D|	D t
	rT_ℓ|	→	D|	D t

Figure 2. Algorithmically generated parser for a single algebraic language.

T t	T|*	*		t (14,2)
	E+T|	→	E|	E t
	+T|	→	E|	E t
	T|	→	E|	E t
E t	E|+	*		t (12,2)
	(E|	*		t (18,3)
	i←E|	→	S|	S t
F t	F↑|	*		t (16,2)
	T*F|	→	T|	T t
	F|	→	T|	T t
P t	F↑P|	→	F|	F t
	P|	→	F|	F t
B t	|-B|	*		t (0,3)
S t	H;S|	→	H|	H t
∑ t				SUCCESS EXIT

Figure 2. (continued)

REFERENCES

1. Earley J. Generating a Recognizer for a BNF Grammar, Carnegie-Mellon Institute of Technology, June 1965, unpublished.

2. Evans, A. An ALGOL 60 Compiler, National ACM Conference, Denver 1963.

3. Feldman, J. A Formal Semantics for Computer-Oriented Languages, Doctoral Thesis, Carnegie-Mellon Institute of Technology, 1964.

4. Floyd, R. A Descriptive Language for Symbol Manipulation, J. ACM 8, 4 (1961), 579-584

5. Floyd, R. Bounded Context Syntactic Analysis, Comm. ACM 7, 2 (1964), 62-67.

6. Knuth, D. On the Translation of Languages from Left to Right, Information and Control 8, (1965), 607-639.

7. Standish, T. Generating Productions from a Restricted Class of BNF Grammars, Carnegie-Mellon Institute of Technology Computation Center, unpublished.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

2a. REPORT SECURITY CLASSIFICATION

UNCLASSIFIED

2b. GROUP

3. REPORT TITLE

ON THE GENERATION OF PARSERS FOR BNF GRAMMARS: AN ALGORITHM

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Research Report

5. AUTHOR(S) *(First name, middle initial, last name)*

Franklin L. DeRemer

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
	19	7

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
46-26-15-305	ILLIAC IV DOCUMENT NO. 199
b. PROJECT NO. US AF 30(602)4144	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* DCS Report No. 276
d.	

10. DISTRIBUTION STATEMENT

Qualified requesters may obtain copies of this report from DCS.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
NONE	Rome Air Development Center Griffiss Air Force Base Rome, New York 13440

13. ABSTRACT

This paper describes an algorithm which, for suitable grammars, maps the Backus Naur Form (BNF) definition of the grammar of a language into a parser for the sentences in that language. By design the algorithm generates a suitable parser for any bounded right context grammar. It happens that it also covers some LR(k) grammars which are not bounded right context. A modified version of Floyd's descriptive language for symbol manipulation is used to describe the parser. Several examples illustrate the application and generality of the algorithm.

DD FORM 1 NOV 68 1473

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Parser						
Syntax analysis						
Compiler						
Compiler-Compiler						